

CLAIMS

What is claimed is:

- 1 1. A computer implemented method of allocating stack memory for a
2 process for executing a computer program code, the method comprising:
- 3 mapping an active session to a thread for execution, the thread having a
4 first stack memory selected to execute a first class of code;
- 5 responsive to a code segment of the code being of the first class,
6 executing the code segment with the first stack memory; and
- 7 responsive to the code segment being of a second class, executing the
8 code segment in an auxiliary stack memory to execute the code segment and reclaiming
9 the auxiliary stack memory subsequent to executing the code segment.
- 1 2. The method of claim 1, wherein the code segment includes a function call and
2 code segments of the second class include a wrapper configured to call the auxiliary
3 stack memory to execute the function call.
- 1 3. The method of Claim 2, wherein the thread is non-preemptive, the auxiliary
2 stack memory is a shared stack, and the wrapper performs the operations of:

- 3 saving a stack pointer to the first stack;
- 4 resetting the stack pointer to the shared stack;
- 5 copying arguments from the first stack to the shared stack;
- 6 calling a program function of the function call;
- 7 returning the result to the first stack of the thread; and
- 8 returning the shared stack.

1 4. The method of claim 2, wherein the thread is preemptive, the auxiliary stack
2 is a new stack from a pool of stacks, and the wrapper performs the operations of:

- 3 saving a stack pointer to the first stack memory;
- 4 allocating a new stack segment having a stack address;
- 5 saving the stack address of the new stack segment;
- 6 resetting the stack pointer to the new stack segment;
- 7 copying an argument from the first stack to the new stack;

8 calling a program function of the function call;

9 returning the result of the program function to the first stack memory; and

10 returning the new stack segment.

1 5. The method of claim 1, further comprising: allocating a preselected stack
2 memory space for the auxiliary stack memory.

1 6. The method of Claim 1, further comprising: allocating the stack memory for
2 the auxiliary stack memory space as required to satisfy the stack memory requirements
3 of the function call.

1 7. The method of claim 1, wherein each of the classes includes a code type that is
2 blockable and a code type that is non-blockable.

1 8. The method of Claim 7, wherein the code types are identified by a naming
2 convention.

1 9. A method of reducing stack memory resources in a computer system that
2 executes concurrent user sessions, the method comprising:

3 mapping an active session having a program code to a thread for execution, the
4 thread having a first stack memory space allocated to the thread selected to handle a first
5 class of function calls;

6 transferring the execution of the program code from the first stack memory to an
7 auxiliary stack memory having a stack memory size greater than the first stack memory
8 responsive to the program code invoking a function call of a second class of function
9 calls that requires a stack memory size greater than that of the first stack memory;

10 executing the function call on the auxiliary stack memory;

11 copying a result of the function call to the first stack memory of the thread; and

12 reclaiming the auxiliary stack memory.

1 10. The method of Claim 9, wherein the auxiliary stack memory is a stack
2 selected from a pool of stacks residing in the memory pool.

1 11. The method of Claim 9, wherein the auxiliary stack memory is a shared
2 stack.

1 12. The method of Claim 9, further comprising: selecting the size of the auxiliary
2 stack memory as a function of a code type of the function call.

1 13. The method of Claim 9, further comprising: wrapping the program code in a
2 wrapper to transfer the execution to the auxiliary stack memory.

1 14. The method of Claim 13, wherein the thread is non-pre-emptive and the
2 wrapper performs the steps of:

3 saving a stack pointer to the first stack memory;

4 resetting the stack pointer to a shared stack;

```

5      copying arguments from the first stack to shared stack;

```

6 calling a function;

7 returning the result of the function to the first stack; and

8 returning the shared stack.

1 15. The method of Claim 13, wherein the thread is pre-emptive and the wrapper
2 performs the steps of:

3 saving a stack pointer to the first stack;

3 identifying function calls of the program code requiring stack memory greater
4 than the stack memory allocated to the thread; and

5 wrapping each function call requiring stack memory greater than that allocated to
6 the thread with a wrapper configured to call an auxiliary stack memory to execute the
7 function call.

1 19. The method of Claim 18, further comprising:

2 selecting the stack memory allocated to the thread sufficient to handle a first
3 class of function calls.

1 20. The method of Claim 19, further comprising the step of: selecting the size of
2 the auxiliary stack memory sufficient to handle a second class of function calls.

1 21. The method of Claim 18, wherein the auxiliary stack memory is a new stack
2 from a memory pool.

1 22. The method of Claim 18, further comprising the step of:

2 forming a shared stack as the auxiliary stack memory.

1 23. The method of Claim 18, wherein the code includes a function call having a
2 recursive algorithm, further comprising:

3 replacing the recursive algorithm with an iterative algorithm performing the
4 same function, whereby the size of the stack required to execute the function is reduced.

1 24. The method of Claim 18, wherein the function call includes a stack-allocated
2 variable and further comprising:

3 replacing the stack allocated variable with a heap allocated variable, whereby the
4 size of the stack required to execute the function is reduced.

1 25. The method of Claim 18, further comprising:

2 identifying a program code segment that blocks substantially longer than other
3 program segments; and

4 replacing the program code segment with program code segment(s) performing
5 the same function but selected to reduce the potential blockage time.

1 26. The method of Claim 25, wherein a supervisory program having a database
2 of program code segments is used to implement the method.

1 27. The method of Claim 18, wherein each function call has a corresponding
2 program code naming convention.

2 characterizing at least one function call by running the function on a real or
3 virtual system to determine if the function is blocking or non-blocking.

1 33. A computer readable medium including program code for execution of a
2 process in a computer system, the computer system having at least one computer thread
3 having a first stack memory having a first stack size allocated to the thread and an
4 alternate stack memory space having a second stack size, the program code comprising:

5 a computer program code having code segments of different code class, the code
6 including a first code class that requires the first stack memory size and a second code
7 class that requires the second stack memory size; and

8 a wrapper wrapping each code segment of the second class configured to transfer
9 execution of the function to the alternate stack memory space.

1 34. A computer system having an operating system for concurrently
2 executing a plurality of user session requests, comprising:

3 a computer program residing in a memory, comprising:

4 a pool of threads, each thread having an associated stack memory having
5 a first stack size;

6 a thread mapper mapping each user session onto one of the threads;

7 an auxiliary stack memory having a second stack size, the second stack
8 size being larger than the first stack size;

9 a program code for executing one of the user sessions, the code including
10 at least one code segment characterized by a code class, the code classes including a first
11 code class that requires the first stack memory size and a second code class that requires
12 the second stack memory size; and

13 a wrapper for each code segment of the second class configured to
14 transfer execution of the function to the auxiliary stack memory.

1 35. A computer thread for executing program code, comprising:

2 a first stack memory associated with the thread for executing a first class
3 of function calls requiring a first stack memory size; and

4 switchable auxiliary stack memory means for executing function calls of
5 second class requiring a stack memory resource greater than the first stack
6 memory and reclaiming the stack memory resource when the function call of the
7 second class is completed.

8